Search-Based Test Case Generation with Model-Based Testing Approach

M.Y. Suhaila
School of Information Technology and Electrical Engineering,
The University of Queensland, Brisbane 4072, Australia.
Email: s.mohdyasin@uq.edu.au

***Abstract*—The rapid development nature of web applications remains an open challenge to the testing community. One particular challenge is producing a small but effective set of test cases that could uncover bugs in the system under test. This research examines the combination of a search-based testing with a model-based testing technique to generate and optimise test cases. The goal of this approach is to produce a set of effective test cases for functional testing of web applications that can achieve satisfying test coverage. Modeling languages used to create a model for web applications are considered. Next, search-based testing techniques are investigated. Fitness functions definitions that could evaluate effective test cases are explored. Initial results of the proposed technique is presented and discussed in order to distinguish measures in improving the proposed technique in terms of effectiveness and coverage.**

Keywords—functional testing; search-based testing; genetic algorithm; web application testing; model-based testing

I. INTRODUCTION

Testing web applications (WAs) has their own interesting challenges. WAs commonly consist of widely distributed components that are written in various programming languages [1]. These programming languages are distinguished into server-side, client-side and database programming languages. The integration of multiple programming languages is designed to provide a dynamic web interface, which changes based on requests processed from the user. Since accessibility is provided all the time, WAs are concurrently used by myriads of users with diverse experience levels. The concurrent access of WAs by its users may sometimes lead to the web application behaving unpredictably [2], triggering unanticipated faults. As WAs are event-driven [3], the user is usually prompted to complete a task in a step-by-step manner. Faults will manifest when these processes are not followed, or when the user fails to provide the expected input prior to submitting their request. These issues need to be considered when testing WAs.

This paper presents a search-based testing (SBT) technique that utilises a model-based testing (MBT) technique to produce effective test cases that provide acceptable testing coverage. The Interaction Flow Modeling Language (IFML) is used to model the system under test (SUT) and subsequently generate the initial test cases. Next, the initial test cases are optimised using a genetic algorithm (GA) in order to improve their effectiveness and coverage. Incorporating a model-based testing technique is useful in realising this goal, since the model represents a simplified representation of the WA's behaviour [4]. The use of an SBT algorithm is beneficial in generating good test cases by evolving and mutating the current set of test cases [5]. But in order to do that, a fitness function that can accurately capture the testing goal is needed [6]. A suitable fitness function for this proposed technique is also presented.

II. RELATED WORK

The related work is divided into three issues: search-based testing, model-based testing and fault seeding.

A. Search-based Testing

According to McMinn, SBT is defined as, "the use of a meta-heuristic optimising search techniques to automate or partially automate a testing task" [6]. SBT is applied in software testing with the purpose of finding an optimal solution for a specific problem. SBT only requires two elements: a representation of the problem and a fitness function that captures the objectives [7]. In software testing, test cases and test data are often chosen as the representation for the testing problem, whereas the testing goal will be translated into a suitable fitness function. The fitness function will guide the search technique towards finding a good solution in a given search space, within a practical time limit [6]. The selected search technique is applied to optimise the given representation to produce new solutions, called offsprings. If these offsprings have a higher fitness value than their predecessor, even after achieving the maximum number of generations, this indicates optimal solutions are found and the optimisation process ceased.

SBT has made major contributions in optimising test data and test cases [5]. To date, GAs have been widely used to optimise test cases [8, 9] and whole test suites [10]. Apart from test optimisation, classifying test cases according to their faults using GAs has also been presented [11]. Since GAs are global algorithm, it is believed that they are capable of handling large search spaces. However, Baudry et al. introduce bacteriologic algorithms when GAs could not rapidly improve the mutation scores of a small solution set in .NET environment [12]. Aside from GAs, contributions of other SBT algorithms were also reported. Yoo et al.

combine human expert analysis using an Analytic Hierarchy Process algorithm and automate an Interleaved Cluster Prioritisation algorithm to prioritise clustered test cases with the intention of achieving higher structural coverage [13]. Next, the rural Chinese postman algorithm has been demonstrated to isolate faulty states in a finite-state machine (FSM) representation of the SUT [14]. All of the above addresses issues in testing non-WAs. Due to the different nature and complexity of WAs [2, 15], the need of a customised SBT approach to address challenges in WAs are inevitable.

So far, four works that propose SBT in testing WAs have been published. An algorithm called HILL which is based on the hill-climbing algorithm is applied to an FSM model of Ajax WAs to extract suites of test cases that have longer state-based sequences [16]. For branch coverage, an SBT tool called SWAT that could automate test data generation and reduce testing effort is proposed [17]. Another tool called WETT optimises whole test suites using an evolutionary algorithm (EA) for statement coverage [18], while in security testing, addressing cross-side scripting issues using GAs are introduced to expose security vulnerabilities [19]. These efforts [17-19] concentrate on achieving structural coverage; they do not extensively consider functional testing. Most of the efforts use low-level test case representations. Since one function in WA might consists of several sequences of states or branches, scrambling these states or branches during optimisation might produce inexecutable offsprings. Even though an attempt has been made to represent test cases at a higher level [18], the results presented are directed towards statement coverage, not towards uncovering faults or bugs in the SUT. Furthermore, optimised test cases show a tendency to bloat. To prevent or limit bloatness, controls in the form of steps towards normalising these test cases is performed. However, applying such controls might limit the diversity of the test cases. It has also been shown that using evolutionary algorithms such as GAs might lead to the generation of a lot of unsuitable test cases over and over again. The crossover and mutation operators if applied improperly might lead to syntactically incorrect test scripts, and only a small number of useful evolved offsprings obtainable after numerous cycles of evolution.

B. Model-Based Testing

A model is an abstraction of the SUT's behavior. In MBT, the model is used to gain understanding of how an application's behaviour is processed e.g. the type of inputs accepted, conditions and the expected outputs [20]. Existing model-based testing efforts for web applications have proposed FSMs [4, 21], statecharts [3], Nmodel [22] and also Atomic Section model [23] as modeling languages. Aside from Nmodel which focused on discovering errors in SUT, the others were aimed towards discovering inconsistencies in transitional paths. However, NModel is formerly meant for programs written in *C#*. When it is implemented in modeling WAs, Nmodel took considerable effort in learning and bulding its test harness.

A number of modeling languages that are developed for WAs are being considered for the proposed technique, namely Interaction Flow Modeling Language (IFML) [24], ReWeb [25], and Internet Application Modeling Language (IAML) [26]. Presently, IFML seems the most suitable choice due to its recognition by the Object Management Group (OMG) and its extensive documentation. IFML is designed to present a description of the WAs behaviour from the perspective of the end user [24]. IFML provides high-level representation of the SUT using simple notations, and its initial learning process is relatively easy. Currently, the feasibility of these modeling languages in functional testing of WAs have not yet been reported.

C. Fault Seeding

Fault seeding is performed by introducing artificial faults inside the SUT. The purpose of seeding faults is to establish the effectiveness of the generated test cases in discovering faults residing in the SUT. To achive confidence, faults that are introduced have to resemble real faults as closely as possible [16]. Mutation operators have been used to mutate the SUT during fault seeding [12]. Faults reproduced based on real bug reports of the SUT have also been presented [16]. Apart from these strategies, faults can also be reproduced based on fault taxonomies. Several fault taxonomies suitable for WAs have been introduced based on investigations of natural faults discovered in most WAs [15, 27-29]. Using one of these fault taxonomies to reproduce faults are feasible since they provide coherent classifications of faults.

III. Research Completed to Date

To ensure that this research achieves its goal, a set of research questions were designed.

A. Research Questions

The following research questions are posed:

- How can GAs increase the effectiveness of the test cases for functional testing of WAs through the optimisation process?
- Does IFML provide a suitable representation for test cases when they are optimised using GA?
- How can the effectiveness of the test cases be defined into a fitness function for the respective GA?

B. Case Study

A case study is required in order to demonstrate the feasibility of the approach. For that purpose, a library management system called *OpenBiblio* is chosen as the SUT. It is developed using PHP as its main programming language. Other programming languages involved in its development are HTML, SQL, JavaScript, CSS and XML [30]. *OpenBiblio* is an event-driven WA that feeds on its user's input to dynamically generate the output. Its dynamic web pages consists of GET and POST methods used to fetch, store and update information relating to book loans and fines in its database. Its methods are organised in classes. *OpenBiblio* serves two main types of user: administrator and library staff, with access levels dynamically controlled by the administrator. It has an active development community and is constantly updated. *OpenBiblio* is considered a suitable candidate based on all the features discussed above.

IV. PROPOSED APPROACH

To answer the research questions, a test case generation technique that utilises a GA is proposed. A MBT technique is used in generating the test cases. After initial considerations (see Section II), IFML is selected as the modeling language for creating a model of the SUT and for the representation of the test cases. The proposed technique begins with generating the initial set of test cases from the IFML model of the SUT. Using fault seedings, faults are introduced into the SUT. Figure 1 illustrates an example of a test case of adding a new bibliography in *OpenBiblio*. It starts with visiting the Cataloging web page, which is represented by the rectangle labelled *View Container*. Adding new bibliography information is chosen from the list of actions in Cataloging, represented by the rectangle called *ViewComponent*. The circled arrow indicates an *Event* produced by the user by clicking the new bibliography link. The thin arrows represent *Navigation Flow* that takes the user to another web page. Next, new bibliography information is submitted. The details of the new bibliography is then displayed in another web page. To enable library members to loan the new bibliography, a copy of the bibliography is then created by submitting new copy information of the respective bibliography.

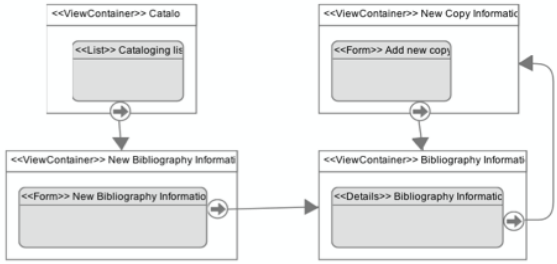

Fig. 1. A test case representation using IFML.

In SBT, the representation of the individual and fitness function are important in determining the sucess of the algorithm [6]. An individual consists of a set of genes. A gene represents either a *ViewComponent,* a *Navigation Flow*, an *Event*, or an *Action* which corresponds to atomic units in IFML. Adopting similar application of GAs [31], the fitness values of initial population of test cases is first measured. In determining the fitness function, similar works were analysed. For now, mutation score is chosen as the fitness function. A pair of individuals are randomly selected from the initial population. These individuals (parents) are evolved using crossover and mutation operators. The crossover phase performs single-point crossover method. The mutation phase requires suitable mutation operators to mutate the individuals. For now, simple swap, insert and delete mutation operators are proposed [18]. These mutation operators are performed with $\frac{1}{3}$ probabilities. This will produce another pair of individuals, called offsprings. The offsprings will be compared with its parents based on their fitness values. Out of the comparative evaluation, the pair of individuals with the highest fitness values is updated into the the current solution set. Next, the current solution set will be measured for its percentage of global mutants that it has killed. This is to ensure that most of the seeded faults are successfully killed. This process is reiterated until an optimal solution set is found (indicated by satisfying percentage of global mutants killed by the set), or until the maximum iteration is reached. If optimal or near-optimal solution set are not found even after maximum iteration is reached, the optimisation process will be reevaluated and refined. Multi-objective fitness function might be considered in the future if it can significantly improves the initial results.

A. Initial Implementation and Results

Modeling of the SUT is achieved using Eclipse 4.2.2 with IFML Editor as its plug-in [32]. A set of 10 test cases representing 10 individuals is generated from the IFML model of the SUT. For now, the test cases are manually generated and optimised using the GA. Prior to this, the SUT is seeded by 12 mutants created based on fault types classified in a fault taxonomy [29]. The initial test cases were then executed and their mutation scores and percentage of killed and alive mutants are noted. There is no equivalent mutants discovered. The first generation reported 75% mutation score and left 25% alive mutants. The mutation score slightly increased on the second iteration (81%) but remains unchanged until the fourth iteration. One of the offsprings produced during the second iteration manage to kill an alive mutant. However, further iterations failed to kill the rest of the alive mutants. Moreever, two of the offsprings were not executable due to invalid navigation flows.

The initial results indicate that GA shows promising result, but further tuning is required. It appears that evolving only a pair of individuals in an iteration is slow and not cost-effective. To overcome this issue, approaches such as evolving whole test suites [33] is a better choice. Another reason is lack of proper controls introduced during the crossover and mutation phases. GA is prone to produce inexecutable test cases if the crossover and mutation phases are not controlled. Exercising certain controls such as allowing only syntactically legitimate crossover and mutation has been proposed [31, 33]. The use of pre-defined configuration files to assist manual modifications on the genes has also been suggested in minimising the risk of producing inexecutable offspring [18]. Another solution is by inspecting the IFML model of the SUT and building a repository of legitimate interaction flows which can be referred to during the crossover and mutation phases. The types of faults introduced during fault seeding may also need to be examined. Since the offsprings fails to kill the remaining alive mutants, a possible reason might be due to the inappropriate choice of fault types e.g. non-functional faults and so on. Until further analysis is performed, these options remain open for consideration.

V. CONTRIBUTIONS

This research outlines the following contributions. Firstly, the novel combination of GA with IFML for testing WAs has never been investigated. Based on the literature review, test case representation using IFML has never been proposed since most efforts emphasise state or branch coverage, as opposed to detecting faults related to WA's behaviour. It is believed that this combination will potentially generate more effective test cases for functional

testing of WAs since IFML could depict a WA's behaviour and its interactions with the front end user. Secondly, the fault seeding process introduces faults based from a fault taxonomy for WAs. These faults were chosen in order to create faults that closely resemble natural faults in WAs. The fitness function is also crucial in determining the success of optimising the inital test cases. Using mutation score as the current fitness values looks encouraging, but it can be improved by rewarding higher values towards test cases that have higher user interaction coverage.

VI. Conclusion and Further Works

The importance of testing WAs is an ongoing issue given the constantly expanding number of WA users. Having smaller but effective test cases could facilitate quicker testing. Aside from that, adequate testing coverage is important to software testers, especially when the need to rapidly develop and deploy WAs is a priority. Introducing a SBT technique with MBT technique could address these issues. The initial implementation shows promising results, but further analysis is required in enhancing the technique in optimising test cases for testing WAs.

Acknowledgment

The author wish to acknowledge the support of the Ministry of Education (MoE) Malaysia in funding this research work under the SLAI scheme.

References

[1] J. Offutt, Y. Wu, X. Du, and H. Huang, "Bypass testing of web applications," in *15th International Symposium on Software Reliability Engineering (ISSRE)*, 2004, pp. 187-197.

[2] M. Benedikt, J. Freire, and P. Godefroid, "VeriWeb: Automatically testing dynamic web sites," in *Proceedings of 11th International World Wide Web Conference (WWW)*, 2002.

[3] H. Reza, K. Ogaard, and A. Malge, "A model based testing technique to test web applications using statecharts," in *Fifth International Conference on Information Technology: New Generations (ITNG)*, 2008, pp. 183-188.

[4] H. Achkar, "Model Based Testing of Web Applications," presented at the The Science Technicians' Association of New Zealand Conference (STANZ), Sydney, Australia, 2010.

[5] P. McMinn, "Search ‐ based software test data generation: a survey," *Software Testing, Verification and Reliability,* vol. 14, pp. 105-156, 2004.

[6] P. McMinn, "Search-based software testing: Past, present and future," in *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2011, pp. 153-163.

[7] M. Harman, P. McMinn, J. T. de Souza, and S. Yoo, "Search based software engineering: Techniques, taxonomy, tutorial," in *Empirical Software Engineering and Verification*, ed: Springer, 2012, pp. 1-59.

[8] A. Seesing and H.-G. Gross, "A genetic programming approach to automated test generation for object-oriented software," presented at the Proceedings of the 1st International Workshop on Evaluation of Novel Approaches to Software Engineering, 2006.

[9] S. Wappler and F. Lammermann, "Using evolutionary algorithms for the unit testing of object-oriented software," in *Proceedings of the 2005 conference on Genetic and evolutionary computation*, 2005, pp. 1053-1060.

[10] G. Fraser and A. Arcuri, "Whole Test Suite Generation," *IEEE Transactions on Software Engineering,* vol. 39, pp. 276-291, 2013.

[11] A. Watkins, E. Hufnagel, D. Berndt, and L. Johnson, "Using genetic algorithms and decision tree induction to classify software failures," *International Journal of Software Engineering and Knowledge Engineering,* vol. 16, pp. 269-291, 2006.

[12] B. Baudry, F. Fleurey, J. M. Jézéquel, and Y. Le Traon, "From genetic to bacteriological algorithms for mutation ‐ based testing," *Software Testing, Verification and Reliability,* vol. 15, pp. 73-96, 2005.

[13] S. Yoo, M. Harman, P. Tonella, and A. Susi, "Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge," in *Proceedings of the 18th international symposium on software testing and analysis*, 2009, pp. 201-212.

[14] Q. Guo, R. M. Hierons, M. Harman, and K. Derderian, "Heuristics for fault diagnosis when testing from finite state machines," *Software Testing, Verification and Reliability,* vol. 17, pp. 41-57, 2007.

[15] A. Marchetto, F. Ricca, and P. Tonella, "Empirical validation of a web fault taxonomy and its usage for fault seeding," in *9th IEEE International Workshop on Web Site Evolution (WSE)*, 2007, pp. 31-38.

[16] A. Marchetto and P. Tonella, "Search-Based Testing of Ajax Web Applications," in *1st International Symposium on Search Based Software Engineering*, 2009, pp. 3-12.

[17] N. Alshahwan and M. Harman, "Automated web application testing using search based software engineering," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, 2011, pp. 3-12.

[18] F. Bolis, A. Gargantini, M. Guarnieri, and E. Magri, "Evolutionary testing of PHP web applications with WETT," in *Search Based Software Engineering*, ed: Springer, 2012, pp. 285-291.

[19] A. Avancini and M. Ceccato, "Security testing of web applications: A search-based approach for cross-site scripting vulnerabilities," in *11th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2011, pp. 85-94.

[20] I. K. El ‐ Far and J. A. Whittaker, "Model ‐ Based Software Testing," *Encyclopedia of Software Engineering*.

[21] A. A. Andrews, J. Offutt, and R. T. Alexander, "Testing web applications by modeling with FSMs," *Software & Systems Modeling,* vol. 4, pp. 326-345, 2005.

[22] J. Ernits, R. Roo, J. Jacky, and M. Veanes, "Model-based testing of web applications using NModel," in *Testing of Software and Communication Systems*, ed: Springer, 2009, pp. 211-216.

[23] J. Offutt and Y. Wu, "Modeling presentation layers of web applications for testing," *Software & Systems Modeling,* vol. 9, pp. 257-280, 2010.

[24] OMG. (Last Accessed, 10 January 2014). *IFML Specification Beta version (OMG document ptc/2013-03-08)*. Available: http://www.omg.org/spec/IFML/

[25] F. Ricca and P. Tonella, "Understanding and restructuring web sites with Reweb," *IEEE Multimedia,* vol. 8, pp. 40-51, 2001.

[26] J. M. Wright, "A Modelling Language for Interactive Web Applications," in *24th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2009, pp. 689-692.

[27] Y. Guo and S. Sampath, "Web application fault classification - An exploratory study," in *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2008, pp. 303-305.

[28] S. Pertet and P. Narasimhan, "Causes of failure in web applications (cmu-pdl-05-109)," *Parallel Data Laboratory,* p. 48, 2005.

[29] N. Mansour and M. Houri, "Testing web applications," *Information and Software Technology,* vol. 48, pp. 31-42, 2006.

[30] B. D. S. Inc. (Last Accessed, 20 November 2013). *The OpenBiblio Open Source Project on Ohloh*. Available: http://www.ohloh.net/p/openbiblio

[31] B. Baudry, F. Fleurey, J.-M. Jézéquel, and Y. Le Traon, "Genes and bacteria for automatic test cases optimization in the. net environment," in *Proceedings of the 13th International Symposium on Software Reliability Engineering (ISSRE)*, 2002, pp. 195-206.

[32] WebRatio. (Last Accessed, 23 November 2013). *IFML Editor*. Available: http://www.webratio.com/portal/content/en/ifml-editor

[33] G. Fraser and A. Arcuri, "Evolutionary generation of whole test suites," in *2011 11th International Conference on Quality Software (QSIC)*, 2011, pp. 31-40.